Compressed Tree Canonization

Markus Lohrey[1], Sebastian Maneth[2], and Fabian Peternek[2]

[1] Universität Siegen, Germany
[2] University of Edinburgh, UK

Abstract. Straight-line (linear) context-free tree (SLT) grammars have been used to compactly represent ordered trees. It is well known that equivalence of SLT grammars is decidable in polynomial time. Here we extend this result and show that isomorphism of unordered trees given as SLT grammars is decidable in polynomial time. The proof constructs a compressed version of the canonical form of the tree represented by the input SLT grammar. The result is generalized to unrooted trees by "re-rooting" the compressed trees in polynomial time. We further show that bisimulation equivalence of unrooted unordered trees represented by SLT grammars is decidable in polynomial time. For non-linear SLT grammars which can have double-exponential compression ratios, we prove that unordered isomorphism is PSPACE-hard and in EXPTIME. The same complexity bounds are shown for bisimulation equivalence.

1 Introduction

Deciding isomorphism between various mathematical objects is an important topic in theoretical computer science that has led to intriguing open problems like the precise complexity of the graph isomorphism problem. An example of an isomorphism problem, where the knowledge seems to be rather complete, is tree isomorphism. Aho, Hopcroft and Ullman [1, page 84] proved that isomorphism of unordered trees (rooted or unrooted) can be decided in linear time. An unordered tree is a tree, where the children of a node are not ordered. The precise complexity of tree isomorphism was finally settled by Lindell [13], Buss [5], and Jenner et al. [11]: Tree isomorphism is logspace-complete if the trees are represented by pointer structures [13,11] and ALOGTIME-complete if the trees are represented by expressions [5,11]. All these results deal with trees that are given explicity (either by an expression or a pointer structure). In this paper, we deal with the isomorphism problem for trees that are given in a succinct way. Several succinct encoding schemes for graphs exist in the literature. Galperin and Wigderson [8] considered graphs that are given by a boolean circuit for the adjacency matrix. Subsequent work showed that the complexity of a problem undergoes an exponential jump when going from the standard input representation to the circuit representation; this phenomenon is known as upgrading, see [7] for more details and references. Concerning graph isomorphism, it was shown in [7] that its succinct version is PSPACE-hard, even for very restricted classes of boolean circuits (DNFs and CNFs).

In this paper, we consider another succinct input representation that has turned out to be more amenable to efficient algorithms, and, in particular, does not show the upgrading phenomenon known for boolean circuits: straight-line context-free grammars,

i.e., context-free grammars that produce a single object. Such grammars have been intensively studied for strings and recently also for trees. Using a straight-line grammar, repeated patterns in an string or tree can be abbreviated by a nonterminal which can be used in different contexts. For strings, this idea is known as grammar-based compression [6,14], and it was extended to trees in [4,16]. In fact this approach can be also extended to general graphs by using hyperedge replacement graph grammars; the resulting formalism is known as hierarchical graph representation and was studied under an algorithmic perspective in [12].

The main topic of this paper is the isomorphism problem for trees that are succinctly represented by straight-line context-free tree grammars. An example of such a grammar contains the productions $S \to A_0(a)$, $A_i(y) \to A_{i+1}(A_{i+1}(y))$ for $0 \leq i \leq n-1$, and $A_n(y) \to f(y, y)$ (here y is called a parameter and in general several parameters may occur in a rule). This grammar produces a full binary tree of height 2^n and hence has $2^{2^n+1} - 1$ many nodes. This example shows that a straight-line context-free tree grammar may produce a tree, whose size is doubly exponential in the size of the grammar. The reason for this double exponential blow-up is copying: The parameter y occurs twice in the right-hand side of the production $A_n(y) \to f(y, y)$. If this is not allowed, i.e., if every parameter occurs at most once in every right-hand side, then the grammar is called linear. Straight-line linear (resp., non-linear) context-free tree grammars are called SLT grammars (resp., ST grammars) in this paper. SLT grammars generalize dags (directed acyclic graphs) that allow to share repeated subtrees of a tree, whereas SLT grammars can also share repeated patterns that are not complete subtrees.

It turned out that many algorithmic problems are much harder for trees represented by ST grammars than trees represented by SLT grammars. A good example is the membership problem for tree automata (PTIME-complete for SLT grammars [17] and PSPACE-complete for ST grammars [15]). A similar situation arises for the isomorphism problem: We prove that

- the isomorphism problem for (rooted or unrooted) unordered trees that are given by SLT grammars is PTIME-complete, and
- the isomorphism problem for (rooted or unrooted) unordered trees that are given by ST grammars is PSPACE-hard and in EXPTIME.

Our polynomial time algorithm for SLT grammars constructs from a given SLT grammar G a new SLT grammar G' that produces a canonical representation of the tree produced by G. Our canonical representation of a given rooted unordered tree t is the ordered rooted tree (in an ordered tree the children of a node are ordered) that has the lexicographically smallest preorder traversal among all ordered versions of t. For unrooted SLT-compressed trees, we first compute a compressed representation of the center node of a given SLT-compressed unrooted tree t. Then we compute an SLT grammar that produces the rooted version of t that is rooted in the center node. This is also the standard reduction of the unrooted isomorphism problem to the rooted isomorphism problem in the uncompressed setting, but it requires some work to carry out this reduction in polynomial time in the SLT-compressed setting.

Our techniques can be also used to show that checking bisimulation equivalence of trees that are represented by SLT grammars is PTIME-complete. This generalizes the well-known PTIME-completeness of bisimulation for dags [2]. In this context, it is

interesting to note that bisimulation equivalence for graphs that are given by hierarchical graph representations is PSPACE-hard and in EXPTIME [3].

2 Preliminaries

For $k \geq 0$ let $[k] = \{1, \ldots, k\}$. Let Σ be an alphabet. By T_Σ we denote the set of all (ordered, rooted) trees over the alphabet Σ. It is defined recursively as the smallest set of strings T such that if $t_1, \ldots, t_k \in T$ and $k \geq 0$ then also $\sigma(t_1, \ldots, t_k)$ is in T. For the tree $a()$ we simply write a. The set $D(t)$ of *Dewey addresses* of a tree $t = \sigma(t_1, \ldots, t_k)$ is the subset of $\mathbb{N}^*$ defined recursively as $\{\varepsilon\} \cup \bigcup_{i \in [k]} i \cdot D(t_i)$. Thus ε denotes the root node of t and $u \cdot i$ denotes the i-th child of u. For $u \in D(t)$, we denote by $t[u] \in \Sigma$ the symbol at u, i.e., if $t = \sigma(t_1, \ldots, t_k)$, then $t[\varepsilon] = \sigma$ and $t[i \cdot u] = t_i[u]$. The *size* of the tree t is $|t| = |D(t)|$.

A *ranked alphabet* N is a finite set of symbols each of which equipped with a non-negative integer, called its "rank". We write $A^{(k)}$ to denote that the rank of A is k, and write $N^{(k)}$ for the set of symbols in N that have rank k. For an alphabet Σ and a ranked alphabet N, we denote by $T_{N \cup \Sigma}$ the set of trees t over $N \cup \Sigma$ with the property that if $t[u] = A \in N^{(k)}$, then $u \cdot i \in D(t)$ if and only if $i \in [k]$. Thus, if a node is labeled by a ranked symbol, then the rank determines the number of children of the node.

We fix a special alphabet $Y = \{y_1, y_2, \ldots\}$ of *parameters*. For y_1 we also write y. The parameters are considered as symbols of rank zero, and by $T_{\Sigma \cup N}(Y)$ we denote the set of trees from $T_{\Sigma \cup N \cup Y}$ where each symbol in Y has rank zero. We write Y_k for the set of parameters $\{y_1, \ldots, y_k\}$. For trees $t, t_1, \ldots, t_k \in T_{\Sigma \cup N}(Y)$ we denote by $t[y_j \leftarrow t_j \mid j \in [k]]$ the tree obtained from t by replacing in parallel every occurrence of y_j ($j \in [k]$) by t_j.

A *context-free tree grammar* is a tuple $G = (N, \Sigma, S, P)$ where N is a ranked alphabet of nonterminal symbols, Σ is an alphabet of terminal symbols with $\Sigma \cap N = \emptyset$, $S \in N^{(0)}$ is the start nonterminal, and P is a finite set of productions of the form $A(y_1, \ldots, y_k) \to t$ where $A \in N^{(k)}$, $k \geq 0$, and $t \in T_{N \cup \Sigma}(Y_k)$. Occasionally, we consider context-free tree grammars without a start nonterminal. Two trees $\xi, \xi' \in T_{N \cup \Sigma}(Y)$ are in the one-step derivation relation $\Rightarrow_G$ induced by G, if ξ has a subtree $A(t_1, \ldots, t_k)$ with $A \in N^{(k)}$, $k \geq 0$ such that ξ' is obtained from ξ by replacing this subtree by $t[y_j \leftarrow t_j \mid j \in [k]]$, where $A(y_1, \ldots, y_k) \to t$ is a production in P. The tree language $L(G)$ produced by G is $\{t \in T_\Sigma \mid S \Rightarrow_G^* t\}$. We assume that G contains no useless productions, i.e., each production as applied in the derivation of some terminal tree in T_Σ. The *size* of the grammar G is $|G| = \sum_{(A(y_1, \ldots, y_k) \to t) \in P} |t|$. The grammar $G = (N, \Sigma, S, P)$ is *deterministic* if for every $A \in N$ there is exactly one production of the form $A \to t$. The grammar G is *acyclic*, if there is a linear order $<$ on N such that $A < B$ whenever B occurs in a tree t with $(A \to t) \in P$. A deterministic and acyclic grammar is called *straight-line*. Note that $|L(G)| = 1$ for a straight-line grammar. We denote the unique tree t produced by the straight-line tree grammar G by $\mathsf{val}(G)$. Moreover, for a tree $t \in T_{\Sigma \cup N}(Y)$ we denote with $\mathsf{val}_G(t) \in T_\Sigma(Y)$ the unique tree obtained from t by applying productions from G until only terminal symbols from Σ occur in the tree. If G is clear from the context, we simply write $\mathsf{val}(t)$

for $\mathsf{val}_G(t)$. The grammar G is *linear* if for every production $(A \to t) \in P$ and every $y \in Y$, y occurs at most once in t.

For a *straight-line linear context-free tree grammar* we say *SLT grammar*. For a (not necessarily linear) *straight-line context-free tree grammar* we say *ST grammar*. Most of this paper is about SLT grammars, only in Section 6 we deal with (non-linear) ST grammars. SLT grammars generalize rooted node-labelled dags (directed acyclic graph), where the tree defined by such a dag is obtained by unfolding the dag starting from the root (formally, the nodes of the tree are the directed paths in the dag that start in the root). A dag can be viewed as an SLT grammar, where all nonterminals have rank 0 (the nodes of the dag correspond to the nonterminal of the SLT grammar). Dags are less succinct than SLT grammars (take the tree $f^N(a)$ for $N = 2^n$), which in turn are less succinct than general ST grammars (take a full binary tree of height 2^n). We need the following fact:

Lemma 1. *A given ST grammar G can be transformed in exponential time into an equivalent SLT grammar.*

Proof. In fact, an ST grammar G can be transformed in exponential time into an equivalent dag. This dag is obtained by viewing the right hand side $t(x_1, \ldots, x_n)$ of a G-production $A(x_1, \ldots, x_n) \to t(x_1, \ldots, x_n)$ as a dag, by merging for all $i \in [k]$ all x_i-labelled leafs into a single x_i-labelled node. In this way, G becomes a so called hyperedge replacement graph grammar (or hierarchical graph definition in the sense of [12]) that produces a dag of exponential size, which can be constructed in exponential time from G, and whose unfolding is $\mathsf{val}(G)$. □

A *context* is a tree in $T_{\Sigma \cup N}(\{y\})$ with exactly one occurrence of y. We denote with $\mathcal{C}_{\Sigma \cup N}$ the set of all contexts and write $\mathcal{C}_\Sigma$ for the set of contexts that contain only symbols from Σ. For a context $t(y)$ and a tree t' we write $t[t']$ for $t[y \leftarrow t']$. Occasionally, we also consider SLT grammars, where the start nonterminal belongs to $N^{(1)}$, i.e., has rank 1. We call such a grammar a 1-*SLT grammar*. Note that $\mathsf{val}(G)$ is a context if G is a 1-SLT grammar G.

In the literature, SLT grammars are usually defined over a ranked terminal alphabets. The following lemma is proved in [17]; the proof immediately carries over to our setting where Σ is not ranked.

Lemma 2. *One can transform in polynomial time an SLT grammar into an equivalent SLT grammar, where each production has one of the following four types (where $\sigma \in \Sigma$ and $A, B, C, A_1, \ldots, A_k \in N$):*

(1) $A \to \sigma(A_1, \ldots, A_k)$,
(2) $A \to B(C)$,
(3) $A(y) \to \sigma(A_1, \ldots, A_i, y, A_{i+1}, \ldots, A_k)$, *or*
(4) $A(y) \to B(C(y))$.

In particular, note that N contains only nonterminals of rank at most 1.

In the following, we will only deal with SLT grammars G having the property from Lemma 2. For $i \in [4]$, we denote with $G(i)$ the SLT grammar (without start nonterminal) consisting of all productions of G of type (i) from Lemma 2.

Region Restrictions. A *straight-line program* (SLP) can be seen as a 1-SLT grammar $G = (N, \Sigma, S, P)$ containing only productions of the form $A(y) \to B(C(y))$ and $A(y) \to \sigma(y)$ with $B, C \in N$ and $\sigma \in \Sigma$. Thus, G contains ordinary rules of a context-free string grammar in Chomsky normal form (but written as monadic trees). Intuitively, if $\mathsf{val}(G) = a_1(\cdots a_n(y)\cdots)$ then G produces the string $a_1 \cdots a_n$ and we also write $\mathsf{val}(G) = a_1 \cdots a_n$. For a string $w = a_1 \cdots a_n$ and two numbers $l, r \in [n]$ with $l \leq r$ we denote by $w[l, r]$ the substring $a_l a_{l+1} \cdots a_r$. The following result is a special case of [9], where it is shown that a so called composition system (an SLP extended with right-hand sides of the form $A[l, r]$ for positions $l \leq r$) can be transformed into an ordinary SLP.

Lemma 3. *For a given SLP G and two binary encoded numbers $l, r \in [|\mathsf{val}(G)|]$ with $l \leq r$ one can compute in polynomial time an SLP G' such that* $\mathsf{val}(G') = \mathsf{val}(G)[l, r]$.

3 Isomorphism of Unrooted SLT-Represented Trees

For a tree t we denote with $\mathsf{uo}(t)$ the unordered rooted version of t. It is the node-labeled directed graph (V, E, λ) where $V = D(t)$ is the set of nodes,

$$E = \{(u, u \cdot i) \mid i \in \mathbb{N}, u \in \mathbb{N}^*, u \cdot i \in D(t)\}$$

is the edge relation, and λ is the node-labelling function with $\lambda(u) = t[u]$. For an SLT grammar G, we also write $\mathsf{val}_{\mathsf{uo}}(G)$ for $\mathsf{uo}(\mathsf{val}(G))$.

In this section, we present a polynomial time algorithm for deciding $\mathsf{uo}(\mathsf{val}(G_1)) = \mathsf{uo}(\mathsf{val}(G_2))$ for two given SLT grammars G_1 and G_2. For this, we will first define a canonical representation of a given tree t, briefly $\mathsf{canon}(t)$, such that $\mathsf{uo}(s)$ and $\mathsf{uo}(t)$ are isomorphic if and only if $\mathsf{canon}(s) = \mathsf{canon}(t)$. Then, we show how to produce for a given SLT grammar G in polynomial time an SLT grammar for $\mathsf{canon}(\mathsf{val}(G))$.

For reasons that will become clear in a moment we have to restrict to trees $t \in T_\Sigma$ that have the following property: For all $u, v \in D(t)$, if $t[u] = t[v]$ then u and v have the same number of children (nodes with the same label have the same number of children). Such trees are called *ranked trees*. For the purpose of deciding the isomorphism problem for unorderd SLT-represented trees this is not a real restriction. Denote for a tree $t \in T_\Sigma$ the ranked tree $\mathsf{ranked}(t)$ such that $D(t) = D(\mathsf{ranked}(t))$ and for every $u \in D(t)$ with $t[u] = \sigma$: if u has k children in t, then $\mathsf{ranked}(t)[u] = \sigma_k$, where σ_k is a new symbol. Then we have:

- $\mathsf{uo}(s)$ and $\mathsf{uo}(t)$ are isomorphic if and only if $\mathsf{uo}(\mathsf{ranked}(s))$ and $\mathsf{uo}(\mathsf{ranked}(t))$ are isomorphic.
- For an SLT grammar G we construct in polynomial time the SLT grammar $\mathsf{ranked}(G)$ obtained from G by changing every production $A \to t$ into $A \to \mathsf{ranked}(t)$, where ranked is extended to trees over Σ and nonterminals by defining $\mathsf{ranked}(t)[u] = t[u]$ if $t[u]$ is a nonterminal. Then we have $\mathsf{val}(\mathsf{ranked}(G)) = \mathsf{ranked}(\mathsf{val}(G))$.

Hence, in the following we will only consider ranked trees, and all SLT grammars will produce ranked trees as well.

3.1 Length-Lexicographical Order and Canons

Let us fix the alphabet Σ. For a tree $t \in T_\Sigma$ we denote by $\mathsf{dflr}(t)$ its depth-first left-to-right traversal string in Σ^*. It is defined as

$$\mathsf{dflr}(\sigma(t_1, \ldots, t_k)) = \sigma\, \mathsf{dflr}(t_1) \cdots \mathsf{dflr}(t_k)$$

for every $\sigma \in \Sigma$, $k \geq 0$, and $t_1, \ldots, t_k \in T_\Sigma$. Note that for ranked trees s and t it holds that: $\mathsf{dflr}(s) = \mathsf{dflr}(t)$ if and only if $s = t$. This is the reason for restricting to ranked trees: for unranked trees this equivalence fails. For instance, $a(a(a))$ and $a(a, a)$ have the same depth-first left-to-right traversal string aaa.

Let $<_\Sigma$ be an order on Σ; it induces the *lexicographical ordering* $<_{\mathsf{lex}}$ on equal-length strings $u, w \in \Sigma^*$ as: $u <_{\mathsf{lex}} w$ if and only if there exist $p, u', w' \in \Sigma^*$ and letters $a, b \in \Sigma$ with $a <_\Sigma b$ such that $u = pau'$ and $w = pbw'$. The *length-lexicographical ordering* $<_{\mathsf{llex}}$ on Σ^* is defined by $u <_{\mathsf{llex}} w$ if and only if (i) $|u| < |w|$ or (ii) $|u| = |w|$ and $u <_{\mathsf{lex}} w$. We extend the definition of $<_{\mathsf{llex}}$ to trees s, t over Σ by $s <_{\mathsf{llex}} t$ if and only if $\mathsf{dflr}(s) <_{\mathsf{llex}} \mathsf{dflr}(t)$.

Lemma 4. *Let G, H be SLT grammars. It is decidable in polynomial time whether or not (1) $\mathsf{val}(G) <_{\mathsf{llex}} \mathsf{val}(H)$ and (2) whether or not $\mathsf{val}(G) = \mathsf{val}(H)$.*

Proof. Point (2) was shown in [4] by computing from G, H in polynomial time SLPs G', H' with $\mathsf{val}(G') = \mathsf{dflr}(\mathsf{val}(G))$ and $\mathsf{val}(H') = \mathsf{dflr}(\mathsf{val}(H))$. Equivalence of SLPs can be decided in polynomial time; this was proved independently in [10,19,20], cf. [14].

To show (1), we compute in two single bottom-up runs the numbers $n_1 = |\mathsf{val}(G')|$ and $n_2 = |\mathsf{val}(H')|$. If $n_1 \neq n_2$ we are done; so assume that $n = n_1 = n_2$. Next, we compute the first position for which the strings $\mathsf{val}(G')$ and $\mathsf{val}(H')$ differ. This is done via binary search and polynomially many equivalence tests: We compute $m = \lceil n/2 \rceil$ and, using Lemma 3, construct SLPs G_1 and G_2 for $\mathsf{val}(G')[1, m]$ and $\mathsf{val}(G')[m{+}1, n]$, respectively, and SLPs H_1 and H_2 for $\mathsf{val}(H')[1, m]$ and $\mathsf{val}(H')[m{+}1, n]$, respectively. We proceed with G_1 and H_1 if $\mathsf{val}(G_1) \neq \mathsf{val}(H_1)$, otherwise we proceed with G_2 and H_2. After $c \leq \lceil \log(n) \rceil$ many steps we obtain SLPs G_c, H_c representing the first position for which $\mathsf{val}(G')$ and $\mathsf{val}(H')$ differ. We compute the terminal symbols g, h with $\mathsf{val}(G_c) = \{g\}$ and $\mathsf{val}(H_c) = \{h\}$ and determine whether or not $g <_\Sigma h$. □

For a tree $t \in T_\Sigma$ we define its *canon* $\mathsf{canon}(t)$ as the smallest tree s with respect to $<_{\mathsf{llex}}$ such that $\mathsf{uo}(s)$ is isomorphic to $\mathsf{uo}(t)$. Clearly, if $\mathsf{canon}(t) = t$ then also $\mathsf{canon}(t') = t'$ for every subtree t' of t. Hence, in order to determine $\mathsf{canon}(t)$ for $t = \sigma(t_1, \ldots, t_k)$ ($\sigma \in \Sigma$, $k \geq 0$) let $c_i = \mathsf{canon}(t_i)$ for $i \in [k]$ and let $c_{i_1} \leq_{\mathsf{llex}} c_{i_2} \leq_{\mathsf{llex}} \ldots \leq_{\mathsf{llex}} c_{i_k}$ be the length-lexicographically ordered list of the canons $c_1, \ldots, c_k$. Then $\mathsf{canon}(t) = \sigma(c_{i_1}, \ldots, c_{i_n})$. The following lemma can be easily shown by an induction on the tree structure:

Lemma 5. *Let $s, t \in T_\Sigma$. Then $\mathsf{uo}(s)$ and $\mathsf{uo}(t)$ are isomorphic if and only if $\mathsf{canon}(s) = \mathsf{canon}(t)$.*

3.2 Canonizing SLT-Represented Trees

In the following, we denote a tree $A_1(A_2(\cdots A_n(t)\cdots))$, where $A_1, A_2, \ldots, A_n$ are unary nonterminals with $A_1A_2\cdots A_n(t)$.

Theorem 6. *From a given SLT grammar G one can construct in polynomial time an SLT grammar G' such that* $\mathsf{val}(G') = \mathsf{canon}(\mathsf{val}(G))$.

Proof. Let $G = (N, \Sigma, S, P)$. We assume that G contains no distinct nonterminals $A_1, A_2 \in N^{(0)}$ such that $\mathsf{val}_G(A_1) = \mathsf{val}_G(A_2)$. This is justified because we can test $\mathsf{val}_G(A_1) = \mathsf{val}_G(A_2)$ in polynomial time by Lemma 4 (and replace A_2 by A_1 in G in such a case). We will add polynomially many new nonterminals to G and change the productions for nonterminals from $N^{(0)}$ such that for the resulting SLT grammar G' we have $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for every $Z \in N^{(0)}$.

Consider a nonterminal $Z \in N^{(0)}$ and let M be the set of all nonterminals in G that can be reached from Z. By induction, we can assume that G already satisfies $\mathsf{val}_G(A) = \mathsf{canon}(\mathsf{val}_G(A))$ for every $A \in M^{(0)} \setminus \{Z\}$. We distinguish two cases.

Case (i). Z is of type (1) from Lemma 2, i.e., has a production $Z \to \sigma(A_1, \ldots, A_k)$. Using Lemma 4 we construct an ordering $i_1, \ldots, i_k$ of $[k]$ such that $\mathsf{val}_G(A_{i_1}) \leq_{\mathsf{llex}} \mathsf{val}_G(A_{i_2}) \leq_{\mathsf{llex}} \cdots \leq_{\mathsf{llex}} \mathsf{val}_G(A_{i_k})$. We obtain G' by replacing the production $Z \to \sigma(A_1, \ldots, A_k)$ by $Z \to \sigma(A_{i_1}, \ldots, A_{i_k})$ and get $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$.

Case (ii). Z is of type (2), i.e., has a production $Z \to B(A)$. Let $\{S_1, \ldots, S_m\} = M^{(0)} \setminus \{Z\}$ be an ordering such that

$$\mathsf{val}_G(S_1) <_{\mathsf{llex}} \mathsf{val}_G(S_2) <_{\mathsf{llex}} \cdots <_{\mathsf{llex}} \mathsf{val}_G(S_m).$$

Note that A is one of these S_i. The sequence $S_1, S_2, \ldots, S_m$ partitions the set of all trees t in T_Σ into intervals $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_m$ with

- $\mathcal{I}_0 = \{t \in T_\Sigma \mid t <_{\mathsf{llex}} \mathsf{val}_H(S_1)\}$,
- $\mathcal{I}_i = \{t \in T_\Sigma \mid \mathsf{val}_H(S_i) \leq_{\mathsf{llex}} t <_{\mathsf{llex}} \mathsf{val}_H(S_{i+1})\}$ for $1 \leq i < m$, and
- $\mathcal{I}_m = \{t \in T_\Sigma \mid \mathsf{val}_H(S_m) \leq_{\mathsf{llex}} t\}$.

Consider the maximal $G(4)$-derivation starting from $B(A)$, i.e.,

$$B(A) \Rightarrow^*_{G(4)} B_1B_2\cdots B_N(A),$$

where B_i is a typ-(3) nonterminal. Clearly, the number N might be of exponential size, but the set $\{B_1, \ldots, B_N\}$ can be easily constructed. In order to construct an SLT for $\mathsf{canon}(\mathsf{val}_G(Z))$, it remains to reorder the arguments in right-hand sides of the type-(3) nonterminals B_i. The problem is of course that different occurences of a type-(3) nonterminal in the sequence $B_1B_2\cdots B_N$ have to be reordered in a different way. But we will show that the sequence $B_1B_2\cdots B_N$ can be split into $m+1$ blocks such that all occurrences of a type-(3) nonterminal in one of these blocks have to be reordered in the same way.

Let $t_k = \mathsf{val}_G(B_kB_{k+1}\cdots B_N(A))$ for $k \in [N]$ and $t_{N+1} = \mathsf{val}_G(A)$. Note that $t_1 = \mathsf{val}_G(Z) >_{\mathsf{llex}} \mathsf{val}_G(S_m)$ and that $t_{k+1} <_{\mathsf{llex}} t_k$ for all k. For $i \in [m]$ let k_i be the

maximal position $k \leq N+1$ such that $t_k \geq_{\mathsf{llex}} \mathsf{val}_G(S_i)$. Since $t_1 \geq_{\mathsf{llex}} \mathsf{val}_G(S_m) \geq_{\mathsf{llex}} \mathsf{val}_G(S_i)$ this position is well defined. Also note that if $A = S_i$, then we have $k_i = k_{i-1} = \cdots = k_1 = N+1$. For every $0 \leq i \leq m$, the interval $[k_{i+1}+1, k_i]$ is the set of all positions k such that $\mathsf{val}_G(t_k) \in \mathcal{I}_i$. Here we set $k_{m+1} = 0$ and $k_0 = N+1$. Clearly, the interval $[k_{i+1}+1, k_i]$ might be empty. The positions $k_0, \ldots, k_m$ can be computed in polynomial time, using binary search combined with Lemma 4. To apply the latter, note that for a given position k we can compute in polynomial time an SLT grammar for the tree t_k using Lemma 3 for the SLP consisting of all type-(4) productions that are used to derive $B_1 B_2 \cdots B_N$.

We now factorize the string $B_1 B_2 \cdots B_N$ as $B_1 B_2 \cdots B_N = u_m u_{m-1} \cdots u_0$, where $u_m = B_1 \cdots B_{k_m - 1}$ and $u_i = B_{k_{i+1}} \cdots B_{k_i - 1}$ for $0 \leq i \leq m-1$. By Lemma 3 we can compute in polynomial time an SLP G_i for the string u_i. For the further consideration, we view G_i as a 1-SLT grammar consisting only of type-(4) productions. Note that $\mathsf{val}(G_i)$ is a linear tree, where every node is labelled with a type-(3) nonterminal. We now add reordered versions of type-(3) productions to G_i. Consider a type-(3) production $(C(y) \to \sigma(A_1, \ldots, A_j, y, A_{j+1}, \ldots, A_k)) \in P$ where $C \in \{B_1, \ldots, B_N\}$. Then we add to G_i the type-(3) production

$$C(y) \to \sigma(A_{j_1}, \ldots, A_{j_\nu}, y, A_{j_{\nu+1}}, \ldots, A_{j_k}),$$

where $\{j_1, \ldots, j_k\} = [k]$ and $0 \leq \nu \leq k$ are chosen such that

(1) $\mathsf{val}_G(A_{j_1}) \leq_{\mathsf{llex}} \mathsf{val}_G(A_{j_2}) \leq_{\mathsf{llex}} \cdots \leq_{\mathsf{llex}} \mathsf{val}_G(A_{j_k})$ and
(2) $\mathsf{val}_G(A_{j_\nu}) \leq_{\mathsf{llex}} \mathsf{val}_G(S_i) <_{\mathsf{llex}} \mathsf{val}_G(A_{j_{\nu+1}})$.

Note that if $\nu = k$ then condition (2) states that $\mathsf{val}_G(A_{j_k}) \leq_{\mathsf{llex}} \mathsf{val}_G(S_i)$, and if $\nu = 0$ then it states that $\mathsf{val}_G(S_i) <_{\mathsf{llex}} \mathsf{val}_G(A_{j_1})$. Also note that condition (2) ensures that for every tree $t \in \mathcal{I}_i$ we have $\mathsf{val}_G(A_{j_\nu}) \leq_{\mathsf{llex}} t <_{\mathsf{llex}} \mathsf{val}_G(A_{j_{\nu+1}})$. Hence, $\mathsf{val}_G(\sigma(A_{j_1}, \ldots, A_{j_\nu}, t, A_{j_{\nu+1}}, \ldots, A_{j_k}))$ is a canon. The crucial observation now is that the above factorization $u_m u_{m-1} \cdots u_0$ of $B_1 B_2 \cdots B_N$ was defined in such a way that for every occurrence of a type-(3) nonterminal $C(y)$ in u_i, the parameter y will be substituted by a tree from $\mathcal{I}_i$ during the derivation from Z to $\mathsf{val}_G(Z)$. Hence, we reorder the arguments in the right-hand sides of nonterminal occurrences in u_i in the correct way to obtain a canon.

We now rename the nonterminals in the SLT grammars G_i (which are now of type (3) and type (4)) so that the nonterminal sets of $G, G_0, \ldots, G_m$ are pairwise disjoint. Let $X_i(y)$ be the start nonterminal of G_i after the renaming. Then we add to the current SLT grammar G the union of all the G_i, and replace the production $Z \to B(A)$ by $Z \to X_m X_{m-1} \cdots X_0(A)$. The construction implies that $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for the resulting grammar G'.

It remains to argue that the above construction can be carried out in polynomial time. All steps only need polynomial time in the size of the current SLT grammar. Hence, it suffices to show that the size of the SLT grammar is polynomially bounded. The algorithm is divided into $|N^{(0)}|$ many phases, where in each phase it enforces $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for a single nonterminal Z. Consider a single phase, where $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ is enforced for a nonterminal Z. In this phase, we (i) change the production for Z and (ii) add new type-(3) and type-(4) productions

to G (the union of the G_i above). But the number of these new productions is polynomially bounded in the size of the initial SLT grammar (the one before the first phase), because the nonterminals introduced in earlier phases are not relevant for the current phase. This implies that the additive size increase in each phase is bounded polynomially in the size of the initial grammar. □

Corollary 7. *The problem of deciding whether* $\mathsf{val}_{\mathsf{uo}}(G_1)$ *and* $\mathsf{val}_{\mathsf{uo}}(G_2)$ *are isomorphic for given SLT grammars* G_1 *and* G_2 *is* PTIME*-complete.*

Proof. Membership in PTIME follows immediately from Lemma 4, Lemma 5, and Theorem 6. Moreover, PTIME-hardness already holds for dags, i.e., SLT grammars where all nonterminals have rank 0, as shown in [18]. □

4 Isomorphism of Unrooted Unordered SLT-Represented Trees

An unrooted unordered tree t over Σ can be seen as a node-labeled (undirected) graph $t = (V, E, \lambda)$, where $E \subseteq V \times V$ is symmetric and $\lambda : V \to \Sigma$. For a node v of t we define the eccentricity $\mathsf{ecc}_t(v) = \max_{u \in V} \delta_t(u, v)$ and the diameter $\varnothing(t) = \max_{v \in V} \mathsf{ecc}_t(v)$, where $\delta_t(u, v)$ denotes the distance from u to v (i.e., the number of edges on the path from u to v in t).

Let $t \in T_\Sigma$ be a rooted ordered tree over Σ and let $t' = \mathsf{uo}(t) = (V, E, \lambda)$ be the rooted unordered tree corresponding to t. The tree $\mathsf{ur}(t') = (V, E \cup E^{-1}, \lambda)$ over Σ is the unrooted version of t'. An unrooted unordered tree t can be represented by an SLT grammar G by forgetting the order and root information present in G. Let $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G) = \mathsf{ur}(\mathsf{uo}(\mathsf{val}(G)))$.

In this section it is proved that isomorphism for unrooted unordered trees t_1, t_2 represented by SLT grammars G_1, G_2, respectively, can be solved in polynomial time with respect to $|G_1| + |G_2|$. We reduce the problem to the (rooted) unordered case that was solved in Corollary 7.

Let $t = (V, E, \lambda)$ be an unordered unrooted tree. A node u of t is called *center node of* t if for all leaves v of t:

$$\delta_t(u, v) \leq (\varnothing(s) + 1)/2.$$

Let $\mathsf{center}(t)$ be the set of all center nodes of t. One can compute the center nodes by deleting all leaves of the tree and iterating this step, until the current tree consists of at most two nodes. These are the center nodes of t. In particular, t has either one or two center nodes. Another characterization of center nodes that is important for our algorithm is via longest paths. Let $p = (v_0, v_1, \ldots, v_n)$ be a longest simple path in t, i.e., $n = \varnothing(t)$. Then the middle points $v_{\lfloor n \rfloor}$ and $v_{\lceil n \rceil}$ (which are identical if n is even) are the center nodes of t. These nodes are independent of the concrete longest path p.

Note that there are two center nodes if and only if $\varnothing(t)$ is odd. Since our constructions are simpler if a unique center node exists, we first make sure that $\varnothing(t)$ is even. Let $\#$ be a new symbol not in Σ. For an unrooted unordered tree t we denote by $\mathsf{even}(t)$ the tree where every pair of edge $(u, v), (v, u)$ is replaced by the edges $(u, v'), (v', v), (v, v'), (v', u)$, where v' is a new node labelled $\#$. Then for an SLT

grammar $G = (N, \Sigma, P, S)$ we let $\mathsf{even}(G) = (N, \Sigma \cup \{\#\}, P', S)$ be the SLT grammar where P' is obtained from P by replacing every subtree $\sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma$, $k \geq 1$, in a right-hand side by the subtree $\sigma(\#(t_1), \ldots, \#(t_k))$. Observe that

- $\mathsf{val}_{\mathsf{ur,uo}}(\mathsf{even}(G)) = \mathsf{even}(\mathsf{val}_{\mathsf{ur,uo}}(G))$,
- $\varnothing(\mathsf{even}(t)) = 2 \cdot \varnothing(t)$ is even, i.e., $\mathsf{even}(t)$ has only one center node, and
- trees t and s are isomorphic if and only if $\mathsf{even}(t)$ and $\mathsf{even}(s)$ are isomorphic.

Since $\mathsf{even}(G)$ can be constructed in polynomial time, we assume in the following that every SLT grammar produces a tree of even diameter and therefore has only one center node. For a tree t of even diameter, we denote with $\mathsf{center}(t)$ its unique center node.

Let $u \in V$. We construct a rooted version $\mathsf{root}(t, u)$ of t, with root node u. We set $\mathsf{root}(t, u) = (V, E', \lambda)$, where $E' = \{(v, v') \in E \mid \delta_t(u, v) < \delta_t(u, v')\}$.

Two unrooted unordered trees t_1, t_2 of even diameter are isomorphic if and only if $\mathsf{root}(t_1, \mathsf{center}(t_1))$ is isomorphic to $\mathsf{root}(t_2, \mathsf{center}(t_2))$. Thus, we can solve in polynomial time the isomorphism problem for unrooted unordered trees represented by SLT grammars G, G' by

(1) determining in polynomial time compressed representations $\tilde{u}_1$ and $\tilde{u}_2$ of $u_1 = \mathsf{center}(\mathsf{val}_{\mathsf{ur,uo}}(G))$ and $u_2 = \mathsf{center}(\mathsf{val}_{\mathsf{ur,uo}}(G'))$, respectively (Section 4.1),
(2) constructing in polynomial time SLT grammars G_1, G_2 such that $\mathsf{val}_{\mathsf{uo}}(G_1) = \mathsf{root}(\mathsf{val}_{\mathsf{ur,uo}}(G), u_1)$ and $\mathsf{val}_{\mathsf{uo}}(G_2) = \mathsf{root}(\mathsf{val}_{\mathsf{ur,uo}}(G'), u_2)$ (Section 4.2), and
(3) testing in polynomial time if $\mathsf{val}_{\mathsf{uo}}(G_1)$ is isomorphic to $\mathsf{val}_{\mathsf{uo}}(G_2)$ (Corollary 7).

4.1 Finding Center Nodes

Let $G = (N, \Sigma, S, P)$ be an SLT grammar. A *G-compressed path* p is a string of pairs $p = (A_1, u_1) \cdots (A_n, u_n)$ such that for all $i \in [n]$, $A_i \in N$, $A_1 = S$, $u_i \in D(t_i)$ is a Dewey address in t_i where $(A_i \to t_i) \in P$, $t_i[u_i] = A_{i+1}$ for $i < n$, and $t_i[u_n] \in \Sigma$. If we omit the condition $t_i[u_n] \in \Sigma$, then p is a partial G-compressed path. Note that by definition, $n \leq |N|$. A partial G-compressed path uniquely represents one particular node in the derivation tree of G, and a G-compressed path represents a leaf of the derivation tree and hence a node of $\mathsf{val}(G)$. We denote this node by $\mathsf{val}_G(p)$. The concatenation $u_1, u_2, \ldots, u_n$ of the Dewey addresses is denoted by $u(p)$.

For a context $t(y) \in \mathcal{C}_\Sigma$ we define $\mathsf{ecc}(t) = \mathsf{ecc}_t(y)$ (recall that in a context there is a unique occurence of the parameter y) and $\mathsf{rty}(t) = \delta_t(\varepsilon, y)$ (the distance from the root to the parameter y). For a tree $s \in T_\Sigma$ we denote with $h(s)$ its height. We extend these notions to contexts $t \in \mathcal{C}_{\Sigma \cup N}$ and trees $s \in T_{\Sigma \cup N}$ by $\mathsf{ecc}(t) = \mathsf{ecc}(\mathsf{val}_G(t))$, $\mathsf{rty}(t) = \mathsf{rty}(\mathsf{val}_G(t))$, and $h(s) = h(\mathsf{val}_G(s))$.

Eccentricity, distance from root to y, and height can be computed in polynomial time for all nonterminals bottom-up. To do so, observe that for two contexts $t(y), t'(y) \in \mathcal{C}_{\Sigma \cup N}$ and a tree $s \in T_{\Sigma \cup N}$ we have

- $\mathsf{rty}(t[t']) = \mathsf{rty}(t) + \mathsf{rty}(t')$,
- $\mathsf{ecc}(t[t']) = \max\{\mathsf{ecc}(t'), \mathsf{ecc}(t) + \mathsf{rty}(t')\}$, and
- $h(t[s]) = \max\{h(s), \mathsf{rty}(t) + h(s)\}$.

Similarly, for a context $t(y) = \sigma(s_1, \dots s_i, y, s_{i+1}, \dots, s_k)$ and a tree $s = \sigma(s_1, \dots, s_k)$ we have:

- $\mathsf{rty}(t) = 1$,
- $\mathsf{ecc}(t) = 2 + \max\{h(s_i) \mid 1 \leq i \leq k\}$, and
- $h(s) = 1 + \max\{h(s_i) \mid 1 \leq i \leq k\}$.

Finally, note that for the tree $t[s]$ ($t(y) \in \mathcal{C}_\Sigma, s \in T_\Sigma$) we have

$$\varnothing(t[s]) = \max\{\varnothing(t), \varnothing(s), \mathsf{ecc}(t) + h(t)\}. \tag{1}$$

Our search for the center node of an SLT-compressed tree is based on the following lemma. For a context $t(y) \in \mathcal{C}_\Sigma$, where u is the Dewey address of the parameter y, and a tree $s \in T_\Sigma$ we say that a node v of $t[s]$ belongs to t if the Dewey address of v is in $D(t) \setminus \{u\}$. Otherwise, we say that v belongs to s, which means that u is a prefix of the Dewey address of v.

Lemma 8. *Let $t(y) \in \mathcal{C}_\Sigma$ be a context and $s \in T_\Sigma$ a tree such that $\varnothing(t[s])$ is even. Let $c = \mathsf{center}(t[s])$. Then we have the following:*

- *If $\mathsf{ecc}(t) \leq h(s)$ then c belongs to s.*
- *If $\mathsf{ecc}(t) > h(s)$ then c belongs to t.*

Proof. Let us first assume that $\mathsf{ecc}(t) \leq h(s)$, Then we have $\varnothing(t) \leq 2 \cdot \mathsf{ecc}(t) \leq \mathsf{ecc}(t) + h(s)$, i.e., $\varnothing(t[s]) = \max\{\varnothing(s), \mathsf{ecc}(t) + h(s)\}$ by (1). Together with $\mathsf{ecc}(t) \leq h(s)$ this implies that the middle point of a longest path in $s[t]$ (which is c) belongs to the tree s.

Next, assume that $\mathsf{ecc}(t) = h(s)+1$. Then we have $\varnothing(s) \leq 2 \cdot h(s) < \mathsf{ecc}(t) + h(s)$, i.e., $\varnothing(t[s]) = \max\{\varnothing(t), \mathsf{ecc}(t) + h(s)\}$. Moreover, we claim that $\mathsf{ecc}(t) + h(s) \geq \varnothing(t)$. In case $\varnothing(t) = \mathsf{ecc}(t)$, this is clear. Otherwise, $\varnothing(t) > \mathsf{ecc}(t)$ and a longest path in t does not end in the parameter node y. It follows that $\varnothing(t) \leq 2 \cdot (\mathsf{ecc}(t) - 1) < \mathsf{ecc}(t) + h(s)$. Thus, we have $\varnothing(t[s]) = \mathsf{ecc}(t) + h(s) = 2 \cdot h(s) + 1$, which is odd, a contradiction. Hence, this case cannot occur.

Finally, assume that $\mathsf{ecc}(t) > h(s)+1$. Again, we get $\varnothing(t[s]) = \max\{\varnothing(t), \mathsf{ecc}(t) + h(s)\}$. Moreover, since $\mathsf{ecc}(t) > h(s) + 1$ the center nodes c must belong to t. □

Lemma 9. *For a given SLT grammar G such that $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G)$ has even diameter, one can construct a G-compressed path for $\mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$.*

Proof. Consider the recursive Algorithm 1. It is started with $t_l = y$, $t_r = p = \varepsilon$ and $A = S$ and computes the node $\mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$. The following invariants are preserved by the algorithm: If $\mathsf{center}(t_l, A, t_r, p)$ is called, then we have:

- If A has rank 0 then $t_r = \varepsilon$
- $\mathsf{val}(G) = \mathsf{val}(t_l[A[t_r]])$ (here we set $t[\varepsilon] = t$).
- The tree $t_l[A[t_r]]$ can be derived from the start variable S.
- p is the partial G-compressed path to the distinguished A in $t_l[A[t_r]]$.
- $\mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$ belongs to the subcontext $\mathsf{val}(A)$ in $\mathsf{val}(t_l)[\mathsf{val}(A)[\mathsf{val}(t_r)]]$.

Algorithm 1 Recursive procedure to find the G-compressed path for the center node

procedure $\mathsf{center}(t_l, A, t_r, p)$
  if $A \to B(C)$ (and thus $t_r = \varepsilon$) or $A(y) \to B(C(y))$ **then**
    if $\mathsf{ecc}(t_l[B(y)]) \leq h(C[t_r])$ **then**
      return $\mathsf{center}(t_l[B(y)], C, t_r, p \cdot (A, 1))$
    else
      return $\mathsf{center}(t_l, B, C[t_r], p \cdot (A, \varepsilon))$
  if $A \to \sigma(A_1, \ldots, A_k)$ (and thus $t_r = \varepsilon$) **then**
    $t_i \leftarrow t_l[\sigma(A_1, \ldots, A_{i-1}, y, A_{i+1}, \ldots, A_k)]$ for all $i \in [k]$
    if there is an $i \in [k]$ such that $\mathsf{ecc}(t_i) \leq h(A_i)$ **then**
      return $\mathsf{center}(t_i, A_i, \varepsilon, p \cdot (A, i))$
    else
      return $(p \cdot (A, \varepsilon))$
  if $A(y) \to \sigma(A_1, \ldots A_{s-1}, y, A_{s+1}, \ldots, A_k)$ **then**
    $t_i \leftarrow t_l[\sigma(A_1, \ldots, A_{i-1}, y, A_{i+1}, \ldots, A_{s-1}, t_r, A_{s+1}, \ldots, A_k)]$ if $i < s$
    $t_i \leftarrow t_l[\sigma(A_1, \ldots, A_{s-1}, t_r, A_{s+1}, \ldots, A_{i-1}, y, A_{i+1}, \ldots, A_k)]$ if $s < i$
    if there is an $i \in [k] \setminus \{s\}$ such that $\mathsf{ecc}(t_i) \leq h(A_i)$ **then**
      return $\mathsf{center}(t_i, A_i, \varepsilon, p \cdot (A_i, i))$
    else
      return $(p \cdot (A, \varepsilon))$

For a call $\mathsf{center}(t_l, A, t_r, p)$, the algorithm distinguishes on the right-hand side of A. If this right-hand side has the form $A(B)$ or $A(B(y))$, then, by comparing $\mathsf{ecc}(t_l[B(y)])$ and $h(C[t_r])$, we determine, whether the search for the center node has to continue in B or C, see Lemma 8.

The case that the right-hand side of A has the form $\sigma(A_1, \ldots, A_k)$ is a bit more complicated. Let $s_l = \mathsf{val}(t_l)$ and $s_i = \mathsf{val}(A_i)$ (by the first invariant we know that $t_r = \varepsilon$). We have to find the center node of $t := s_l(\sigma(s_1, \ldots, s_k)$ and by the last invariant we know that it is contained in $\sigma(s_1, \ldots, s_k)$. We now consider all k many cuts of t along one of the edges between the σ-node and one of the s_i, i.e., we cut t into $s_l(\sigma(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_k)$ and s_i. Using again Lemma 8, it suffices to compare $\mathsf{ecc}(s_l(\sigma(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_k)))$ and $h(s_i)$ in order to determine whether the center node belongs to $s_l(\sigma(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_k)$ or s_i. If for some i, it turns out that the center node is in s_i, then we continue the search with A_i. Finally, assume that for all i, it turns out that the center node is in $s_l(\sigma(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_k)$. Since by the last invariant, the center node is in $\sigma(s_1, \ldots, s_k)$, the σ-labelled node must be the center node. The case of a production $A(y) \to \sigma(A_1, \ldots A_{s-1}, y, A_{s+1}, \ldots, A_k)$ can be dealt with similarly.

Note that $|t_l| + |t_r|$ stays bounded by the size of G. Hence, whenever $\mathsf{ecc}(t)$ and $h(t)$ have to be determined by the algorithm, then t is a polynomial size tree build from terminal and nonterminal symbols. By the previous remarks, $\mathsf{ecc}(t)$ and $h(t)$ can be computed in polynomial time. □

4.2 Re-Rooting of SLT Grammars

Let $G = (N, \Sigma, S, P)$ be an SLT grammar (as usual, having the normal form from Lemma 2) and p a G-compressed path. Let $s(p) \in T_{\Sigma \cup N}$ be the tree defined inductively as follows: Let $(A \to t) \in P$ and $u \in D(t)$. Then $s((A, u)) = t$. If $p = (A, t)p'$ with p' non-empty, then either (i) $u = \varepsilon$ and $t = B(C)$ or (ii) $u = i \in \mathbb{N}$ and $t[i] \in N^{(0)}$. In case (i) we set $s(p) = s(p')[C]$, in case (ii) we set $s(p) = t'[s(p')]$, where $t'(y)$ is obtained from t by replacing the i-th argument of the root by y. Note that $s(p') \in \mathcal{C}_{\Sigma \cup N}(\{y\})$ if p' starts with a nonterminal of rank 1. Let $s = s(p)$; its size is bounded by the size of G. Note that $s[u(p)]$ is a terminal symbol (recall that $u(p)$ denotes the concatenation of the Dewey addresses in p). Assume that $s[u(p)] = \sigma \in \Sigma$. Let $\#$ be a fresh symbol and let s' be obtained from s by changing the label at $u(p)$ from σ to $\#$. Let $s' \Rightarrow_G^* s''$ be the shortest derivation such that $s''[\varepsilon] = \delta \in \Sigma$ (it consists of at most $|N|$ derivation steps). We denote the $\#$-labeled node in s'' by u. Finally, let t be obtained from s'' by changing the unique $\#$ into σ. We define the *p-expansion* of G, denoted $\mathsf{ex}_G(p)$, as the tuple (t, u, σ, δ). Note that $\mathsf{val}_G(p)$ is the unique $\#$-labelled node in $\mathsf{val}_G(s'')$. Moreover, the p-expansion can be computed in polynomial time from G and p.

The p-expansion (t, u, σ, δ) has all information needed to construct a grammar G' representing the rooted version at p of $\mathsf{val}(G)$. If $u = \varepsilon$ then also $\mathsf{val}_G(p) = \varepsilon$. Since G is already rooted at ε nothing has to be done in this case and we return $G' = G$. If $u \neq \varepsilon$ then $\mathsf{val}_G(p) \neq \varepsilon$ and hence t contains two terminal nodes which uniquely represent the root node and the node $\mathsf{val}_G(p)$ of the tree $\mathsf{val}(G)$.

Let $s_1 \in T_\Sigma$ be a rooted ordered tree representing the unrooted unordered tree $\tilde{s}_1 = \mathsf{ur}(\mathsf{uo}(s_1))$. Let $u \neq \varepsilon$ be a node of s_1. Let $s_1[\varepsilon] = \delta \in \Sigma$ and $s_1[u] = \sigma \in \Sigma$. A rooted ordered tree s_2 that represents the rooted unordered tree $\tilde{s}_2 = \mathsf{root}(\tilde{s}_1, u)$ can be defined as follows: Since $u \neq \varepsilon$, we can write

$$s_1 = \delta(\zeta_1, \ldots, \zeta_{i-1}, t'[\sigma(\xi_1, \ldots, \xi_m)], \zeta_{i+1}, \ldots, \zeta_k),$$

where t' is a context, and $u = iu'$, where u' is the Dewey address of the parameter y in t'. We can define s_2 as

$$s_2 = \sigma(\xi_1, \ldots, \xi_m, \mathsf{rooty}(t')[\delta(\zeta_1, \ldots, \zeta_{i-1}, \zeta_{i+1}, \ldots, \zeta_k)]),$$

where rooty is a function mapping contexts to contexts defined recursively as follows, where $f \in \Sigma$, $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_\ell \in T_\Sigma$, and $t(y), t'(y) \in \mathcal{C}_\Sigma$:

$$\mathsf{rooty}(y) = y \tag{2}$$

$$\mathsf{rooty}(f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_\ell)) = f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_\ell) \tag{3}$$

$$\mathsf{rooty}(t[t'(y)]) = \mathsf{rooty}(t')[\mathsf{rooty}(t(y))] \tag{4}$$

Intuitively, the mapping rooty unroots a context $t(y)$ towards its y-node u, i.e., it reverses the path from the root to u. Thus, for instance, $\mathsf{rooty}(f(a, y, b)) = f(a, y, b)$ and $\mathsf{rooty}(f(a, g(c, y, d), b)) = g(c, f(a, y, b), d)$.

Lemma 10. *From a given SLT grammar G and a G-compressed path p one can construct in polynomial time an SLT grammar G' such that $\mathsf{val}_{\mathsf{uo}}(G')$ is isomorphic to $\mathsf{root}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G), \mathsf{val}_G(p))$.*

Proof. Let $G = (N, \Sigma, S, P)$ and $\mathsf{ex}_G(p) = (t, u, \sigma, \delta)$. If $u = \varepsilon$ then define $G' = G$. If $u \neq \varepsilon$ then we can write

$$t = \delta(B_1, \ldots, B_{i-1}, t'[\sigma(\xi_1, \ldots, \xi_m)], B_{i+1}, \ldots, B_k), \tag{5}$$

where $B_j \in N^{(0)}$, $\xi_j \in T_N$, t' is a context composed of nonterminals $A \in N^{(1)}$ and contexts $f(\zeta_1, \ldots, \zeta_{j-1}, y, \zeta_{j+1}, \ldots, \zeta_l)$ ($f \in \Sigma$, $\zeta_j \in T_N$), and $u = iu'$, where u' is the Dewey address of the parameter y in t'.

We define $G' = (N \uplus N', \Sigma, S, P')$ where $N' = \{A' \mid A \in N^{(1)}\}$. To define the production set P', we extend the definition of rooty to contexts from $\mathcal{C}_{\Sigma \cup N}$ by (*i*) allowing in the trees t_j from Equation (3) also nonterminals, and (*ii*) defining for every $B \in N^{(1)}$, $\mathsf{rooty}(B(y)) = B'(y)$. We now define the set of productions P' of P as follows: We put all productions from P except for the start production $(S \to s) \in P$ into P'. For the start variable S we add to P' the production

$$S \to \sigma(\xi_1, \ldots, \xi_m, \mathsf{rooty}(t')[\delta(B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_k)]).$$

Moreover, let $A \in N^{(1)}$ and $(A(y) \to \zeta) \in P$. If this is a type-(3) production, then we add $A'(y) \to \zeta$ to P'. If $\zeta = B(C(y))$ then add $A'(y) \to C'(B'(y))$ to P'.

Claim: Let $A \in N^{(1)}$. Then $\mathsf{val}_{G'}(A') = \mathsf{rooty}(\mathsf{val}_G(A))$.

The claim is easily shown by induction on the reverse hierarchical structure of G: Let $(A \to t_A) \in P$. If $t_A = f(A_1, \ldots, A_j, y, A_{j+1}, \ldots, A_l)$ then $\mathsf{rooty}(\mathsf{val}_G(A)) = \mathsf{val}_G(A)$. Since $(A' \to t_A) \in P'$ and G' contains all productions of G except for the start production, we obtain $\mathsf{val}_{G'}(A') = \mathsf{rooty}(\mathsf{val}_G(A))$. If $t_A = B(C(y))$ then, by Equation (4), $\mathsf{rooty}(\mathsf{val}_G(B(C(y)))) = \mathsf{rooty}(\mathsf{val}_G(C))[\mathsf{val}_G(B)]$. By induction the latter is equal to $\mathsf{val}_{G'}(C')[\mathsf{val}_{G'}(B')]$ which equals $\mathsf{val}(A')$ by the definition of the right-hand side of A'. This proves the claim.

The above claim implies that $\mathsf{val}_{G'}(\mathsf{rooty}(c(y))) = \mathsf{rooty}(\mathsf{val}_G(c(y)))$ for every context $c(y)$ that is composed of contexts $f(\zeta_1, \ldots, \zeta_{j-1}, y, \zeta_{j+1}, \ldots, \zeta_l)$ ($\zeta_j \in T_N$) and nonterminals $A \in N^{(1)}$. In particular, $\mathsf{val}_{G'}(\mathsf{rooty}(t')) = \mathsf{rooty}(\mathsf{val}_G(t'(y)))$ for the context t' from Equation (5). Hence, with $s_j = \mathsf{val}_{G'}(\xi_j) = \mathsf{val}_G(\xi_j)$ and $t_j = \mathsf{val}_{G'}(B_j) = \mathsf{val}_G(B_j)$ we obtain

$$\begin{aligned}
\mathsf{val}(G') &= \mathsf{val}_{G'}(\sigma(\xi_1, \ldots, \xi_m, \mathsf{rooty}(t')[\delta(B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_k)])) \\
&= \sigma(s_1, \ldots, s_m, \mathsf{val}_{G'}(\mathsf{rooty}(t'))[\delta(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_k)]) \\
&= \sigma(s_1, \ldots, s_m, \mathsf{rooty}(\mathsf{val}_G(t'))[\delta(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_k)]).
\end{aligned}$$

Since $\mathsf{val}(G) = \delta(t_1, \ldots, t_{i-1}, \mathsf{val}_G(t')[\sigma(s_1, \ldots, s_m)], t_{i+1}, \ldots, t_k)$, it follows that $\mathsf{val}_{\mathsf{uo}}(G')$ is isomorphic to $\mathsf{root}(\mathsf{val}_{\mathsf{ur,uo}}(G), \mathsf{val}_G(p))$. □

Corollary 11. *The problem of deciding whether* $\mathsf{val}_{\mathsf{ur,uo}}(G_1)$ *and* $\mathsf{val}_{\mathsf{ur,uo}}(G_2)$ *are isomorphic for given SLT grammars* G_1 *and* G_2 *is* PTIME*-complete.*

Proof. The upper bound follows from Lemma 9, Lemma 10, and Corollary 7. Hardness for PTIME follows from the PTIME-hardness for dags [18] and the fact that isomorphism of rooted unordered trees can be reduced to isomorphism of unrooted unordered trees by labelling the roots with a fresh symbol. □

5 Bisimulation on SLT-compressed trees

Fix a set Σ of node labels. Let $G = (V, E, \lambda)$ be a directed node-labelled graph, i.e., $E \subseteq V \times V$ is the edge relation and $\lambda : V \to \Sigma$ is the labelling function. A binary relation $R \subseteq V \times V$ is a bisimulation on G, if for all $(u, v) \in R$ the following three conditions hold:

- $\lambda(u) = \lambda(v)$
- If $(u, u') \in E$, then there exists $v' \in V$ such that $(v, v') \in E$ and $(u', v') \in R$.
- If $(v, v') \in E$, then there exists $u' \in V$ such that $(u, u') \in E$ and $(u', v') \in R$.

Let the relation $\sim$ be the union of all bisimulations on G. It is itself a bisimulation (and hence the largest bisimulation) and an equivalence relation. Two rooted unordered trees s, t with node labels from Σ and roots r_s, r_t are bisimulation equivalent if $r_s \sim r_t$ holds in the disjoint union of s and t. For instance, the trees $f(a, a, a)$ and $f(a, a)$ are bisimulation equivalent but the trees $f(g(a), g(b))$ and $f(g(a, b))$ are not.

For a rooted unordered tree t we define the bisimulation canon $\mathsf{bcanon}(t)$ inductively as follows: Let $t = f(t_1, \ldots, t_n)$ ($n \geq 0$) and let $b_i = \mathsf{bcanon}(t_i)$. Let $s_1, \ldots, s_m$ be a list of trees such that (i) for every $i \in [m]$, s_i is isomorphic to one of the b_j, and (ii) for every $i \in [n]$ there is a unique $j \in [m]$ such that s_i and b_j are isomorphic as rooted unordered trees. Then $\mathsf{bcanon}(t) = f(s_1, \ldots, s_m)$. In other words: Bottom-up, we eliminate repeated subtrees among the children of a node. For instance, $\mathsf{bcanon}(f(a, a, a)) = f(a) = \mathsf{bcanon}(f(a, a))$. The following lemma can be shown by a straightforward induction on the height of trees.

Lemma 12. *Let s and t be rooted unordered trees. Then s and t are bisimulation equivalent if and only if $\mathsf{bcanon}(s)$ and $\mathsf{bcanon}(t)$ are isomorphic.*

The proof of the following theorem is similar to those of Theorem 6.

Theorem 13. *From a given SLT grammar G one can compute a new SLT grammar G' such that $\mathsf{val}_{\mathsf{uo}}(G')$ is isomorphic to $\mathsf{bcanon}(\mathsf{val}_{\mathsf{uo}}(G))$.*

Proof. Let $G = (N, \Sigma, S, P)$. We will add polynomially many new nonterminals to G and change the productions for nonterminals from $N^{(0)}$ such that for the resulting SLT grammar G' we have $\mathsf{uo}(\mathsf{val}_{G'}(Z)) = \mathsf{bcanon}(\mathsf{uo}(\mathsf{val}_G(Z)))$ for every $Z \in N^{(0)}$.

Consider a nonterminal $Z \in N^{(0)}$ and let M be the set of all nonterminals in G that can be reached from Z. By induction, we can assume that G already satisfies $\mathsf{uo}(\mathsf{val}_G(A)) = \mathsf{bcanon}(\mathsf{uo}(\mathsf{val}_G(A)))$ for every $A \in M^{(0)} \setminus \{Z\}$. Moreover, we can assume that G contains no distinct nonterminals $A_1, A_2 \in N^{(0)}$ such that $\mathsf{uo}(\mathsf{val}_G(A_1))$ and $\mathsf{uo}(\mathsf{val}_G(A_2))$ are isomorphic. This is justified because by Corollary 7 we can test in polynomial time whether $\mathsf{uo}(\mathsf{val}_G(A_1))$ and $\mathsf{uo}(\mathsf{val}_G(A_2))$ are isomorphic and replace A_2 by A_1 in G in such a case (the tree produced by the new grammar is isomorphic to $\mathsf{uo}(\mathsf{val}(G))$). Similarly, if there is a type-(1) production $A \to \sigma(A_1, \ldots, A_k)$ such that $A_i = A_j$ for $i < j$, then we remove A_j from the parameter list, and the same is done for type-(3) productions. These preprocessing steps do not change the bisimulation canon. We now distinguish two cases.

Case (i). Z is of type (1), i.e., has a production $Z \to \sigma(A_1, \ldots, A_k)$. By the above preprocessing, we already have $\mathsf{uo}(\mathsf{val}_G(Z)) = \mathsf{bcanon}(\mathsf{uo}(\mathsf{val}_G(Z)))$, so nothing has to be done.

Case (ii). Z is of type (2), i.e., has a production $Z \to B(A)$. For $C \in M^{(0)}$ let $n_C = |\mathsf{val}_G(C)|$ and let

$$J = \{n_C \mid C \in M^{(0)} \setminus \{Z\}\}.$$

We can compute this set of numbers easily in a bottom-up fashion.

Consider the maximal $G(4)$-derivation starting from $B(A)$, i.e.,

$$B(A) \Rightarrow^*_{G(4)} B_1 B_2 \cdots B_N(A),$$

where B_i is a typ-(3) nonterminal. Let $t_k = \mathsf{val}_G(B_k B_{k+1} \cdots B_N(A))$ for $k \in [N]$ and $t_{N+1} = \mathsf{val}_G(A)$. For a given position we can compute in polyomial time the size $|t_i|$ by first computing an SLT grammar for t_i and then computing the size of the generated tree bottom-up. Clearly, the sequence $|t_1|, |t_2|, \ldots, |t_{N+1}|$ is monotonically decreasing. This allows to compute, using binary search, the set of positions

$$I = \{i \mid i \in [N+1], |t_i| \in J\}.$$

Note that $|I| \leq |M^{(0)} \setminus \{Z\}|$ and $N+1 \in I$. Next, we check in polynomial time, using Corollary 7, for every position $i \in I$, whether $\mathsf{uo}(t_i)$ is isomorphic to $\mathsf{uo}(\mathsf{val}_G(C))$ for some $C \in M^{(0)} \setminus \{Z\}$. If such a j exists then we keep i in the set I, otherwise we remove i from I. After this step, I contains exactly those positions $i \in I$ such that $\mathsf{uo}(t_i)$ is isomorphic to $\mathsf{uo}(\mathsf{val}_G(C))$ for some $C \in M^{(0)} \setminus \{Z\}$.

Assume that $I = \{i_1, \ldots, i_k\}$ with $1 \leq i_1 < i_2 < \cdots < i_{k-1} < i_k = N+1$. We now factorize the string $B_1 B_2 \cdots B_N$ as

$$B_1 B_2 \cdots B_N = u_1 B_{i_1 - 1} u_2 B_{i_2 - 1} \cdots u_k B_{i_k - 1},$$

where $u_j = B_{i_{j-1}} \cdots B_{i_j - 2}$ for $j \in [k]$ (set $i_0 = 1$). By Lemma 3 we can compute in polynomial time an SLP G_j for the string u_j. Moreover, we can compute the nonterminals $B_{i_j - 1}$ in polynomial time. For the further consideration, we view G_j as a 1-SLT grammar consisting only of type-(4) productions. Note that $\mathsf{val}(G_j)$ is a linear tree, where every node is labelled with a type-(3) nonterminal.

We now rename the nonterminals in the SLT grammars G_j so that the nonterminal sets of $G, G_1, \ldots, G_k$ are pairwise disjoint. Let $X_j(y)$ be the start nonterminal of G_j after the renaming. Then we add to the current SLT grammar G the union of all the G_j. Moreover, for every $j \in [k]$ we add a new nonterminal C_j to G, whose right-hand side is derived from the right-hand side of $B_{i_j - 1}$ as follows: Let the right-hand side for $B_{i_j - 1}$ be $\sigma(A_1, \ldots, A_l, y)$ (we can assume that the parameter occurs at the last argument position, since this is not relevant for the bisimulation canon). We now check whether there exists an A_i ($i \in [l]$) such that $\mathsf{uo}(\mathsf{val}_G(A_i))$ is isomorphic to $\mathsf{uo}(t_{i_j})$. If such an i exists then by our preprocessing it is unique, and we add to G the production $C_j(y) \to \sigma(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_l, y)$. If such an i does not exist, then the new

nonterminal C_j is not needed. In order to keep the notation uniform, let $C_j = B_{i_j-1}$. Finally, we redefine the production for Z to

$$Z \to X_1 C_1 X_2 C_2 \cdots X_k C_k(A).$$

This concludes the construction of the SLT grammar G'. As in the proof of Theorem 6 one can argue that the size of G' is polynomially bounded in the size of G. □

From Corollary 7, Lemma 12, and Theorem 13 we get:

Theorem 14. *For given SLT grammars G_1 and G_2 one can check in polynomial time, whether $\mathsf{val}_{\mathsf{uo}}(G_1)$ and $\mathsf{val}_{\mathsf{uo}}(G_2)$ are bisimulation equivalent.*

6 Unordered Isomorphism of Non-Linear ST Grammars

In this section, we consider ST grammars that are not necessarily linear.

Theorem 15. *The question, whether $\mathsf{val}_{\mathsf{uo}}(G_1)$ and $\mathsf{val}_{\mathsf{uo}}(G_2)$ are isomorphic for two given ST grammars G_1 and G_2 is* PSPACE*-hard and in* EXPTIME.

Proof. The upper bound follows from Lemma 1 and Corollary 7. For the lower bound, we use a reduction from QBF. Recall that the input for QBF is a quantified boolean formula of the form

$$\Psi = Q_1 z_1 Q_2 z_2 \cdots Q_n z_n : \varphi(z_1, \ldots, z_n), \tag{6}$$

where $Q_i \in \{\forall, \exists\}$, the z_i are boolean variables, and $\varphi(z_1, \ldots, z_n)$ is a quantifier-free boolean formula. We can assume that in φ, negations only occur in front of variables. We use a reduction from the evaluation problem for boolean expressions to the isomorphism problem for explicitly given rooted unordered trees from [11]. Let us take trees s_1, s_2, t_1, t_2. Consider the two trees s and t in Figure 1 that are built up from s_1, s_2, t_1, t_2. Clearly, $s \cong t$ (s and t are isomorphic) if and only if $s_1 \cong t_1$ and $s_2 \cong t_2$. Similarly, for the trees s and t from Figure 2 we have $s \cong t$ if and only if $s_1 \cong t_1$ or $s_2 \cong t_2$.

Fix the ranked alphabet $\Sigma = \{f, a, b, 0, 1\}$. We will construct a non-linear ST grammar G (without start variable), which contains for every subformula $\psi(v_1, \ldots, v_m)$ (where $\{v_1, \ldots, v_m\} \subseteq \{z_1, \ldots, z_n\}$ is the set of free variables of ψ) of the formula Ψ from (6), two nonterminals $A_\psi(v_1, \ldots, v_m)$ and $B_\psi(v_1, \ldots, v_m)$ such that for all truth values $c_1, \ldots, c_m \in \{0, 1\}$: $\psi(c_1, \ldots, c_m)$ evaluates to 1 if and only if $\mathsf{val}_G(A_\psi)[v_i \leftarrow c_i \mid i \in [m]]$ and $\mathsf{val}_G(B_\psi)[v_i \leftarrow c_i \mid i \in [m]]$ are isomorphic as rooted unordered trees.

The base case is that of a literal z or $\neg z$. We introduce the following productions:

$$A_z(z) \to f(z, 1), \quad B_z(z) \to f(1, z), \quad A_{\neg z}(z) \to f(z, 0), \quad B_{\neg z}(z) \to f(0, z).$$

Now let $\psi(v_1, \ldots, v_m) = \psi_1(x_1, \ldots, x_k) \wedge \psi_2(y_1, \ldots, y_l)$ be a subformula of the quantifier-free part $\varphi(z_1, \ldots, z_n)$ in (6), where $\{v_1, \ldots, v_m\} = \{x_1, \ldots, x_k, y_1, \ldots, y_l\}$.

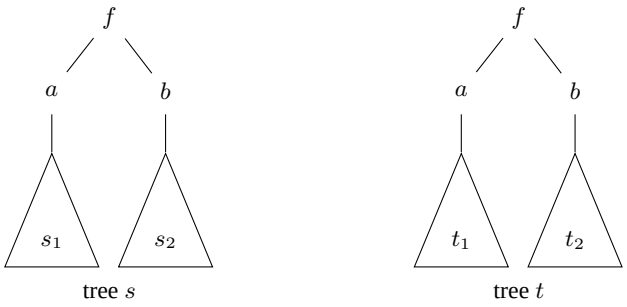

Fig. 1. The and-gadget

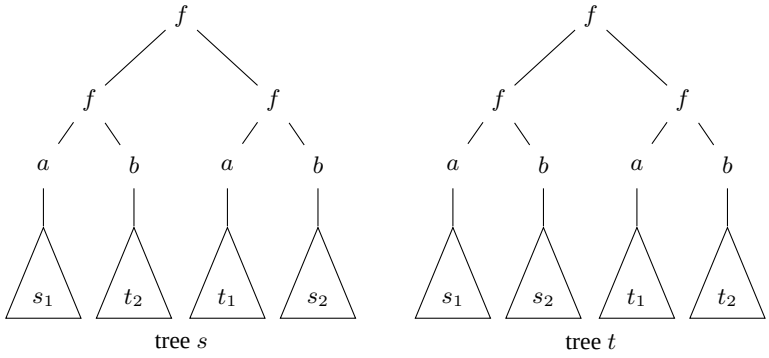

Fig. 2. The or-gadget

Then we use the and-gadget from Figure 1 and set (where $\bar{v} = (v_1, \ldots, v_m)$ and similarly for $\bar{x}$ and $\bar{y}$)

$$A_\psi(\bar{v}) \to f(a(A_{\psi_1}(\bar{x})), b(A_{\psi_2}(\bar{y}))) \text{ and}$$
$$B_\psi(\bar{v}) \to f(a(B_{\psi_1}(\bar{x})), b(B_{\psi_2}(\bar{y}))).$$

If $\psi(v_1, \ldots, v_m) = \psi_1(x_1, \ldots, x_k) \vee \psi_2(y_1, \ldots, y_l)$ then we use the or-gadget from Figure 2 and set

$$A_\psi(\bar{v}) \to f(f(a(A_{\psi_1}(\bar{x})), b(B_{\psi_2}(\bar{y}))), f(a(B_{\psi_1}(\bar{x})), b(A_{\psi_2}(\bar{y})))) \text{ and}$$
$$B_\psi(\bar{v}) \to f(f(a(A_{\psi_1}(\bar{x})), b(A_{\psi_2}(\bar{y}))), f(a(B_{\psi_1}(\bar{x})), b(B_{\psi_2}(\bar{y})))).$$

For a quantified subformula $\psi(z_1, \ldots, z_{i-1}) = \forall z_i\, \psi'(z_1, \ldots, z_{i-1}, z_i)$, we can define the productions similarly (let $\bar{z} = (z_1, \ldots, z_{i-1})$):

$$A_\psi(\bar{z}) \to f(a(A_{\psi'}(\bar{z}, 0)), b(A_{\psi'}(\bar{z}, 1))) \text{ and}$$
$$B_\psi(\bar{z}) \to f(a(B_{\psi'}(\bar{z}, 0)), b(B_{\psi'}(\bar{z}, 1))).$$

Finally, for $\psi(z_1, \ldots, z_{i-1}) = \exists z_i\, \psi'(z_1, \ldots, z_{i-1}, z_i)$ we set

$$A_\psi(\bar{z}) \to f(f(a(A_{\psi'}(\bar{z}, 0)), b(B_{\psi'}(\bar{z}, 1))), f(a(B_{\psi'}(\bar{z}, 0)), b(A_{\psi'}(\bar{z}, 1)))) \text{ and}$$
$$B_\psi(\bar{z}) \to f(f(a(A_{\psi'}(\bar{z}, 0)), b(A_{\psi'}(\bar{z}, 1))), f(a(B_{\psi'}(\bar{z}, 0)), b(B_{\psi'}(\bar{z}, 1)))).$$

This concludes the construction of the ST grammar G. Let $G = (N, \Sigma, P)$. Then we define the two ST grammars $G_1 = (N, \Sigma, A_\Psi, P)$ and $G_2 = (N, \Sigma, B_\Psi, P)$. We have $\mathsf{val}_{\mathsf{uo}}(G_1) \cong \mathsf{val}_{\mathsf{uo}}(G_2)$ if and only if the formula Ψ is true. □

The complexity bounds from Theorem 15 also hold if we want to check whether the unrooted unordered trees $\mathsf{val}_{\mathsf{ur,uo}}(G_1)$ and $\mathsf{val}_{\mathsf{ur,uo}}(G_2)$ are isomorphic: Membership in EXPTIME follows from Lemma 1 and Corollary 11. For PSPACE-hardness, one can take the reduction from the proof of Theorem 15 and label the roots of the final trees with a fresh symbol. Finally, the above PSPACE-hardness proof can be also used for the bisimulation equivalence problem for trees given by ST grammars (the gadgets from Figure 1 and 2 can be reused). Hence, bisimulation equivalence for trees given by ST grammars is PSPACE-hard and in EXPTIME. Since an ST grammar can be transformed into a hierarchical graph definition for a dag (see the proof of Lemma 1), we rediscover the following result from [3]: Bisimulation equivalence for dags that are given by hierarchical graph definitions is PSPACE-hard and in EXPTIME.

7 Open problems

The obvious remaining open problem is the precise complexity of the isomorphism problem for unordered trees that are given by ST grammars. Theorem 15 leaves a gap from PSPACE to EXPTIME. Another interesting open problem is the isomorphism problem for graphs that are given by hierarchical graph definitions. To the knowledge of the authors, this problem has not been studied so far.

References

1. A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison–Wesley, Reading, MA, 1974.
2. J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.
3. R. Brenguier, S. Göller, and O. Sankur. A comparison of succinctly represented finite-state systems. In M. Koutny and I. Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2012.
4. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4–5):456–474, 2008.
5. S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Kurt Gödel Colloquium 97*, pages 18–33, 1997.
6. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
7. B. Das, P. Scharpfenecker, and J. Torán. Succinct encodings of graph isomorphism. In *LATA*, pages 285–296, 2014.
8. H. Galperin and A. Wigderson. Succinct representations of graphs. *Inform. Contr.*, 56:183–198, 1983.
9. C. Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.
10. Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theor. Comput. Sci.*, 158(1&2):143–159, 1996.
11. B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.
12. T. Lengauer and K. W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *J. Comput. Syst. Sci.*, 44:63–93, 1992.
13. S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC*, pages 400–404, 1992.
14. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
15. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoret. Comput. Sci.*, 363(2):196–210, 2006.
16. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013.
17. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.
18. M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Inf. Comput.*, 224:71–105, 2013.
19. K. Mehlhorn, R. Sundar, and C. Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.
20. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470, 1994.